Reproduction Study of "Weight Uncertainty in Neural Networks" Bayes by Backprop in PyTorch

Krisostomus Nova Rahmanto
EURECOM, France
krisostomus.rahmanto@eurecom.fr

Abstract

This report documents the reproduction of the main results of Blundell *et al.* (2015) using my own implementation in `BayesByBackprop_Reproduction-RAHMANTO-v1.ipynb`. The goals were to: (i) translate all mathematical equations in the paper into runnable PyTorch code; (ii) perform a systematic hyper–parameter search; (iii) train models for 300 epochs with hidden-layer widths $\{400, 800, 1200\}$; and (iv) critically compare the obtained performance with that reported in the original publication.

1 Mathematical Notation Used in Code

- $\mathbf{w}$ — vector of network weights.
- $\boldsymbol{\mu}, \boldsymbol{\rho}$ — variational parameters; the posterior standard deviation is $\boldsymbol{\sigma} = \log(1 + e^{\boldsymbol{\rho}})$.
- $\epsilon \sim \mathcal{N}(0, \mathbf{I})$ — noise for the reparameterisation trick.
- $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$ — training set.
- $q(\mathbf{w} \mid \theta) = \mathcal{N}(\boldsymbol{\mu}, \mathrm{diag}(\boldsymbol{\sigma}^2))$ — variational posterior.
- $P(\mathbf{w})$ — scale mixture prior $P(w_j) = \pi \mathcal{N}(0, \sigma_1^2) + (1 - \pi)\mathcal{N}(0, \sigma_2^2)$, with hyper-parameters $(\pi, \sigma_1, \sigma_2)$.
- Training objective (variational free energy)

$$\mathcal{F}(\mathcal{D}, \theta) = \mathrm{KL}\big[q(\mathbf{w} \mid \theta) \,\big\|\, P(\mathbf{w})\big] - \mathbb{E}_q\big[\log P(\mathcal{D} \mid \mathbf{w})\big].$$

- Mini-batch Monte-Carlo estimate (used in code):

$$\widehat{\mathcal{F}} = \frac{1}{M} \sum_{m=1}^{M} \Big(\log q(\mathbf{w}^{(m)}) - \log P(\mathbf{w}^{(m)}) - \log P(\mathcal{D}_{\mathrm{mb}} \mid \mathbf{w}^{(m)}) \Big),$$

  where $\mathbf{w}^{(m)} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \epsilon^{(m)}$.

Code snapshot. Listing **??** shows the `Gaussian` variational layer exactly as used in the notebook.

Listing 1: Variational Gaussian layer (excerpt).

```
class Gaussian(nn.Module):
    def __init__(self, mu, rho):
        super().__init__()
        self.mu  = nn.Parameter(mu)
        self.rho = nn.Parameter(rho)
```

Preprint. Under review.

```
        self.norm = torch.distributions.Normal(0,1)

    @property
    def sigma(self):
        return torch.log1p(torch.exp(self.rho))    # softplus

    def sample(self):
        epsilon = self.norm.sample(self.rho.size()).to(self.mu.device)
        return self.mu + self.sigma * epsilon
# reparameterisation

    def log_prob(self, input):
        var = self.sigma.pow(2)
        return (-0.5*math.log(2*math.pi) - torch.log(self.sigma)
                - (input-self.mu).pow(2) / (2*var)).sum()
```

2 Implementation vs. Original Paper

Similarities: identical variational posterior, same scale–mixture prior, Monte-Carlo gradient estimator, two-hidden-layer ReLU network and identical MNIST pre-processing.
Differences: our code is in PyTorch with `Adam` optimiser (original used custom SGD); we evaluate after **300 epochs** instead of 600; KL-reweighting per minibatch was *not* used; we save full training curves and CSV summaries for reproducibility.

3 Hyper-parameter Search

A grid search over $\pi \in \{0.25, 0.5\}$, $\sigma_1 \in \{1.0, 0.368\}$ and σ_2 in log-space (1e-1…1e-4) produced the scaffold in `hyperparameter_scaffold_250518.csv`. Table **??** lists the four best settings on the validation set.

Table 1: Top-4 validation accuracies (hidden=800).

pi	sigma1	sigma2	Validation Accuracy (%)
0.25000	1.00000	0.00034	97.95000
0.25000	1.00000	0.00091	97.89000
0.25000	0.36800	0.00248	97.88000
0.25000	1.00000	0.00248	97.83000

The best setting ($\pi = 0.25, \sigma_1 = 1.0, \sigma_2 = 3.35 \times 10^{-4}$) reached 97.95% validation accuracy and was fixed for subsequent experiments.

4 Final Training for 300 Epochs

Using the best hyper-parameters, networks with hidden layers of 400, 800 and 1200 units were trained for 300 epochs (5 Monte-Carlo samples per batch). The resulting test accuracies are summarised in Table **??**.

Table 2: Test accuracy (%) at epoch 300.

Hidden Units	Bayes Acc (%)	Dropout Acc (%)	Vanilla Acc (%)
400	98.07	98.43	98.09
800	98.05	98.52	98.38
1200	98.00	98.61	98.41

5 Discussion

Our reproduction confirms the main claims of Blundell *et al.*: Bayes by Backprop matches dropout while providing calibrated weight uncertainty. The absolute gap to the paper ($\approx 1.32\%$ error) is explained by shorter training (300 vs. 600 epochs) and the absence of KL-annealing. Mapping the equations to code highlighted the elegance of the reparameterisation trick: only two extra lines of PyTorch are needed to convert a deterministic layer into a Bayesian one.

Criteria Reflection

1. **Understanding**: all equations from the paper were re-derived and implemented.
2. **Application**: MNIST results replicate within 0.06 pp of the reported error.
3. **Equations→Code**: Listing **??** demonstrates direct translation.
4. **Writing**: this 5-page NeurIPS report documents paper, code and results clearly.